ICSE 97 Doctoral Consortium

Michal Young
Department of Computer Sciences
Purdue University
1398 Computer Science Building
West Lafayette, IN 47907-1398 USA
+1 317 494 6023
young@cs.purdue.edu

ORGANIZING COMMITTEE

Jo Atlee, University of Waterloo, Canada
Simon Kaplan, University of Queensland, Australia
Gail Murphy, University of British Columbia, Canada
Tetsuo Tamai, University of Tokyo, Japan
Michal Young (chair), Purdue University, USA

INTRODUCTION

The ICSE 97 Doctoral Consortium is a one day workshop prior to the regular ICSE technical conference. The goal of the doctoral consortium is to publicly discuss research goals, methods, and results at an early enough stage in Ph.D. research to provide useful guidance in completion of the dissertation research and initiation of a research career. The consortium and ICSE also provide an opportunity for student participants to interact with established researchers and others in the wider software engineering community. Ten student participants were selected from among thirty-nine complete submissions. Selection criteria included the stage of the research and its appropriateness to ICSE, in addition to technical strength of the submitted research abstract.

PARTICIPANTS

Student participants in the Doctoral Consortium are listed below with brief summaries of their research topics. Full research abstracts and contact information are available through the ICSE 97 web site.

Mark Astley, University of Illinois at Urbana-Champaign, USA

Customizable Software Architectures for Distributed Systems

Architectural context refers to the relationship between components of a software architecture and architecture-wide properties such as connectivity and resource usage. Open distributed systems, by their nature, impose complex contexts on their corresponding architectures. The need to ensure properties such as fault-tolerance and consistency, as well as the potential for unpredictable interactions requires modular, dynamically customizable abstractions with composable semantics. To address this need, we model software components as scalable collections of Actors. A meta-architecture provides a modular representation of individual actor contexts. Architectural configuration is specified using first-class connectors which reflectively customize the architectural context of component actors. Moreover, connectors are dynamically composable allowing for graceful evolution of distributed architectures.

ICSE 97 Boston MA USA

Nancy Day, University of British Columbia, Canada

Formal Validation of System Specifications

The goal of this work is to create a framework that systematically maps formal specifications into operational models for analysis. My thesis hypothesis is that explicit operational semantics provide a way to do state-space exploration analysis of model-oriented specifications. This approach offers the advantage that the structure of the specification is captured and can be exploited to reduce the size of the state space in techniques like model checking. It also allows us to analyze models consisting of integrated components specified in different notations, and to return results at the level of abstraction of the specification.

Li Li, George Mason University, USA

Applying Logic-based Databases to Impact Analysis of Object-Oriented Software

We are beginning to see legacy object-oriented systems. An emerging challenge is how to maintain these objects in large, complex systems. Although objects are more easily identified and packaged, features like inheritance make the ripple effects of object-oriented systems more difficult to control than in procedural systems. The research presented here attempts to solve the problem of change-impact analysis of object-oriented software by applying algorithmic software analysis techniques to compute transitive closure of certain relationships among software components. The information about the software components and algorithms can be described in a set of logic rules.

Fernando Brito e Abreu, Universidade Tecnica de Lisboa, Portugal

Object-Oriented Design Metrics

The adoption of the Object-Oriented paradigm is expected to help produce better and cheaper software. The main structural mechanisms of this paradigm, namely, inheritance, encapsulation, information hiding and polymorphism, are the keys to foster reuse and achieve easier maintainability. However, the use of language constructs that support those

mechanisms can be more or less intensive, depending mostly on the designer's ability. In fact, the analysis to design transition is an activity which offers several degrees of liberty. Decisions on best alternatives are usually fuzzy and mostly based on expert judgment. We can then expect rather different quality products to emerge, as well as different productivity gains. Advances in quality and productivity need to be correlated with the use of those constructs. We then need to evaluate this use quantitatively (using design metrics) to guide the OO design process, for instance by means of design heuristics.

Robert DeLine, Carnegie Mellon University, USA
Easing Systems Integration by Overcoming and Avoiding Packaging Mismatch

The packaging mismatch problem remains a significant barrier to the construction of large software systems from off-the-shelf parts. The term packaging refers to the assumptions a software component makes about how it interacts with other components. Components to be integrated suffer from packaging mismatch when each makes assumptions about interaction that the others cannot fulfil. As an example, integrating a Unix filter with a Corba object is problematic: the filter expects to interact through streams; the object, through method calls. In my dissertation research, I will classify the known techniques for overcoming packaging mismatch, both to organize our knowledge and to advise practitioners. Further, I will explore a new approach to avoiding packaging mismatch that separates a component's computational and interaction concerns, so that decisions about interaction can be delayed until integration time.

Jane Hayes, George Mason University, USA
Input Validation Testing: A System Level, Early Lifecycle Technique

Input validation testing is defined as choosing test data that attempt to show the presence or absence of specific faults pertaining to input tolerance. Syntax-directed software accepts inputs from the user, constructed and arranged properly, that control the flow of the application. A large amount of syntax-directed software currently exists and will continue to be developed that should be subjected to input validation testing. System level testing techniques that currently address this area are not well developed or formalized. Input validation testing techniques have not been developed or automated to assist in static input syntax evaluation and test case generation. This thesis will address the problem of statically analyzing input command syntax and then generating test cases for input validation testing, early in the life cycle.

Frank Houdek, University of Ulm, Germany
Development of a Supporting System for Reuse of Software Engineering Experience

Building up and reusing domain specific experience is an essential task in the context of systematic quality improvement programs. As the task of building experience cannot be performed by the software projects, an independent organization unit has to do this, the so-called Experience Factory. To keep the Experience Factory effective, even if people are joining or leaving or the amount of experience grows, there is a demand, first to structure experience and second to provide assistance in handling the experience packages. In this research abstract I propose an approach that helps to fulfill these demands. Main elements of the this approach are formalization of experience packages to enable automatic support, assistance in generalization and aggregation to manage even large numbers of experience packages, and dynamic distance calculation to handle varying environments and to identify key factors for special kinds of experience.

Lamia Labed Jilani, University of Tunis II, Tunisia
Retrieving Software Components by Minimizing Functional Distance

Given a software library used for the purpose of software reuse and whose components are represented by formal specifications, we consider a user query K that no component of the library satisfies. Approximate retrieval consists of identifying the library components that come closest (in some sense) to satisfying query K. In this work, we attempt to discuss what it means for a component to come closest to a specification ; we do so by defining measures of functional distance between specifications, and devising algorithms that minimize these measures over a structured set of components.

Luis G. Nakano, University of Virginia, USA
Integrated Approach to Software Safety

The problem that we propose to study is the integration of software components into system fault-tree analysis. Such analysis is essential in the development of safe systems yet current methods do not handle software well. There are two synergistic techniques that we propose to exploit: (1) The use of results from other engineering areas to design software so that certain properties relevant to the application are met. (2) The use application-domain properties to guarantee that the system will be safe even if certain defects are present in the software. The combination of several approaches will provide engineers with a comprehensive analysis technique.

John Penix, University of Cincinnati, USA
Automated Component Retrieval and Adaptation Using Formal Specifications

This work describes a method for applying formal specifications to automate a system design process based on reusable components and architectures. The focus is on identification and retrieval of components pertinent to a problem, and selection and application of structures (architectures) available for adapting these components. Component retrieval is facilitated by a heuristic based on specification semantics for approximating specification matches that indicate component reusability. To support adaptation, a formal model of architectures is developed that uses algebraic theories to specify relationships between the system and component specifications. Adaptation is performed by modifying or replacing components within an architecture theory.